

Deep Search Labs

News Harbor

BENAZZOU Adnane

Project Documentation:

Data Scraping, Analysis and Web Application Deployment

Table of Contents

Project Documentation:	1
Data Scraping, Analysis and Web Application Deployment	1
I. Introduction	4
II. Data Scraping, cleaning, analysis, and Integration	4
A. Data Scraping	4
B. Data Cleaning	7
C. Data Analysis	8
D. Data Integration	8
III. Backend Setup using FastAPI	9
A. Rationale	9
B. Structure	9
IV. Frontend Setup using React and D3.js	12
V. CI Pipelines Setup using GitHub Actions	15
VI. Conclusion	16

Table of Figures

Figure 1: Data Engineering Tasks Workflow	4
Figure 2: S3 Bucket Folders.....	5
Figure 3: BBC Website Menu	5
Figure 4: BBC Website Submenu.....	5
Figure 5: BBC Website Secondary Layout	6
Figure 6: BBC first paging system.....	6
Figure 7: BBC second paging system	6
Figure 8: Authors Format 1	7
Figure 9: Authors Format 2	7
Figure 10: Authors Format 3	7
Figure 11: MongoDB uploaded files collection.....	8
Figure 12: MongoDB Indexing.....	9
Figure 13: Backend folders structure.....	10
Figure 14: Backend detailed folder structure	11
Figure 15: MongoDB Aggregation pipeline	12
Figure 16: React project structure	13
Figure 17: React components	14

I. Introduction

In an era dominated by information, harnessing and analyzing data has become a pivotal aspect of decision-making and innovation. Additionally, the rapid evolution of technology has enabled us to access an overwhelming amount of information, particularly through online news platforms

The project in our hands, titled "News Harbor," represents a comprehensive project to leverage the volume of data that is increasingly overwhelming day by day. This will be done through implementing a systematic approach which includes the following primary objectives:

- Data Collection and Scraping
- Data Cleaning and Analysis
- Backend Infrastructure
- Frontend Development
- Continuous Integration (CI) Pipelines.

The present document is a documentation that provides a comprehensive overview of each phase of the project, detailing the methods and considerations involved.

II. Data Scraping, cleaning, analysis, and Integration

This integrated phase encapsulates a series of interconnected steps, each contributing to the overarching goal of turning raw data into a coherent and valuable resource.

The below figure encapsulates the general workflow for our process.



Figure 1: Data Engineering Tasks Workflow

A. Data Scraping

The journey begins with the extraction of data from the BBC website, a process orchestrated through Apache Airflow. Employing a combination of Selenium for dynamic content and Requests

for static content, the scraping mechanism ensures a comprehensive collection of up-to-date news articles.

To start off, our task mentioned that if we had no data, we would like to start off with a 3-months historical limit, otherwise just daily data. For this purpose, we have set up an aws s3 Bucket to represent our staging area.



<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class
<input type="checkbox"/>	 clean-data/	Folder	-	-	-
<input type="checkbox"/>	 raw-data/	Folder	-	-	-

Figure 2: S3 Bucket Folders

We considered the raw-data folder only for this step. If any .csv object is detected in the bucket folder, the historical limit is set to the current date, otherwise it is set to a delta limit of 3 months from the current run date.

1. Selenium

Starting off with selenium, in this step we try to make the most of our website, we are aiming to gather news articles from all the available menus and submenus:

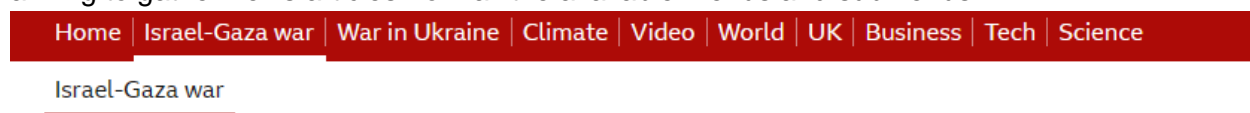


Figure 3: BBC Website Menu

For this purpose, Selenium mimics the user and opens the home page, then stores references to the menu objects as we go over them iteratively. In case of staleness, the page is refreshed, and references are updated.

Following up, we open each Menu on a new tab and once again store the references to its submenus as we go through them iteratively as well. The same logic was applied for both cases while taking into consideration both website layouts:



Figure 4: BBC Website Submenu

Figure 5: BBC Website Secondary Layout

Once this step is reached, Selenium got only few more things to do:

- Gathering Articles Links taking into consideration the historical limit.
- Browsing to the next page.
- Closing the tabs that are no longer used.

2. Requests

You might be wondering why combine both tools. This was done to optimize our extraction speed. While selenium mimics human interaction with web pages, it is rather slow as you cannot parallelize tasks nor open a plethora of pages for inspection.

We could not rely on requests only either. This was since one of BBC layout was using dynamic paging that didn't involve explicit mentioning of the current page:

Latest Updates

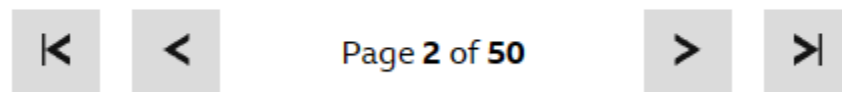


Figure 6: BBC first paging system



Figure 7: BBC second paging system

As we had for each page the full list of articles that we needed to iterate, we dispatched requests in parallel abusing the power of threading. The main objectives here were reaching the article page and gathering all the necessary data we needed: Context (Menu and submenu), title, subtitle, authors, etc.

Finally, the collected data is formatted as a data frame, saved as a csv file and uploaded to the raw data folder.

It is worthy to note that this hybrid approach was a significant improvement over the sole use of selenium where we had to wait over 3 to 4 hours for an unfinished process. Meanwhile this approach took around 40 minutes for the 3 months collection and 3 to 5 minutes for the daily one. A downside for this approach would be that we are not able to collect video links as they require

human interaction to simulate the “play” button. However, as the BBC platform uses streaming services for videos, the gathered links are not available to rehost the videos elsewhere and are thus useless.

B. Data Cleaning

Following the scraping process, the obtained data undergoes a meticulous cleaning procedure. Irrelevant information is filtered out, inconsistencies are rectified, and the dataset is refined to ensure accuracy and coherence.

When initiated, the data cleaning DAG fetches all the uncleaned data thus far from S3. This is done through retrieving objects' names under the raw data folder and checking for their existence in the clean data folder. The delta is then returned for cleaning.

Data Cleaning considered several things, to note mainly:

- Authors text processing: our authors data was rather a mixed one such as:

By Alfred Lasteck in Morogoro and Aaron Akinyemi in London
BBC News

Figure 8: Authors Format 1

By Kelly Ng & Tessa Wong
in Singapore

Figure 9: Authors Format 2

By Lucy Williamson
BBC News, Al-Shuhada, West Bank

Figure 10: Authors Format 3

Whereas, what we needed is a simple list of the authors. The followed strategy was as follow:

- ❖ For any inexistent article authors, we return N/A.
 - ❖ For any existing article authors paragraphs retrieve the first sentence.
 - ❖ If the sentence starts with “By “, remove it.
 - ❖ The subsequent string is then processed to replace all the concatenation strings with a coherent one i.e. ‘,’, ‘and’ are replaced with ‘&’.
 - ❖ The result is then split based on our ‘&’ operator and each author's full name is then gathered.
- Dropping Duplicate URIs: despite tracking our visited and unvisited articles, some of our data was duplicated as it was scraped on parallel from the same page or visited from a different submenu.

- Dropping Data without full text: Our focus is to gather articles data. As such any article page without full text is meaningless.
- Filling Null cells with N/A.

The resulting data from the above process is then uploaded to our s3 clean data bucket.

C. Data Analysis

With a cleaned dataset in hand, the next step involves delving into the realm of data analysis.

This step is document within our analysis notebook which can be found at:

[News-Harbor/Analysis at main · ABenazzou/News-Harbor \(github.com\)](#)

D. Data Integration

The final stride in this comprehensive phase is the integration of the analyzed and refined data. Establishing a robust backend infrastructure using MongoDB.

As our approach uses incremental data scraping and loading, A collection was created on our database to manage this mechanism, storing any already processed file in the database. This is to ensure idempotency in our database.

As such, whenever we want to integrate the clean data from our s3 bucket, we check if it was already uploaded. If not, the file is then considered on the order of their upload date.

```
_id: ObjectId('65a083a58f4bd758ebafc5dc')
file_name: "clean-data/BBC_DATA_2024-01-09_cleaned.csv"
```

```
_id: ObjectId('65a083af8f4bd758ebafc5dd')
file_name: "clean-data/BBC_DATA_2024-01-10_cleaned.csv"
```

Figure 11: MongoDB uploaded files collection

The data integration does not add up any null attribute for storage efficiency leveraging the MongoDB schemeless nature. Additionally, to ensure idempotency within the acquired records (possible overlap between csv files), an upsert mechanism is set in place through the update of existing records while inserting only the new ones.

Finally, As the news harbor project relies heavily on MongoDB, we considered indexing our most used fields for data querying to leverage mongo DB queries and pipelines:

Name and Definition	Type	Size
> _id_	REGULAR ⓘ	327.7 KB
> categories_1	REGULAR ⓘ	49.2 KB
> date_posted_1	REGULAR ⓘ	106.5 KB
> full_text_text	TEXT ⓘ	26.1 MB
> subcategory_1	REGULAR ⓘ	61.4 KB
> topics_1	REGULAR ⓘ	233.5 KB

Figure 12: MongoDB Indexing

This allows us to mitigate growing data issues and speed up response time through avoiding full document scans.

III. Backend Setup using FastAPI

For the News Harbor project, we selected FastAPI as our backend framework. FastAPI is a modern, fast web framework for building APIs based on standard Python type hints. This section outlines our rationale for choosing FastAPI, and the structure of our API.

A. Rationale

For the development phase, we had the choice between 2 popular stacks, FARM (Fast API React MongoDB) and MERN (MongoDB Express.js React Node.js). Below are the main points that encouraged the use of FastAPI over Express.js or Nest.js.

- **Performance, Speed and Data Handling:** FastAPI is known as one of the fastest python web frameworks available. Its asynchronous programming capabilities makes it more efficient for handling large numbers of simultaneous connections which is crucial for our data intensive application. While Node.js is also efficient in handling asynchronous operations, Python's extensive libraries and simplicity in handling data gives FastAPI the edge on this side.
- **Development Efficiency:** FastAPI provides features such as automatic serialization and documentation. This significantly speeds up our development process while express.js requires more manual setup for features such as data validation and documentation.

B. Structure

Our project structure was organized following the below pattern:

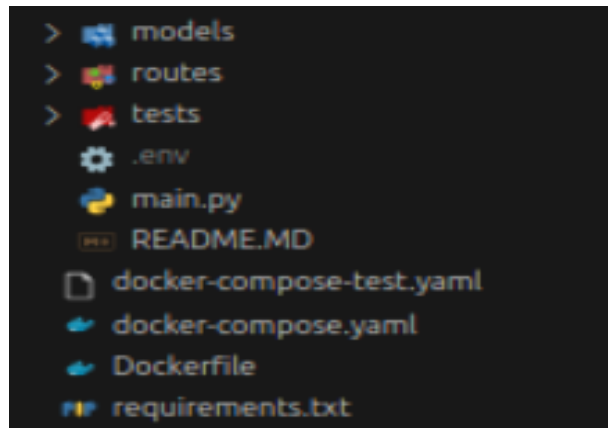


Figure 13: Backend folders structure

- Models' directory contains data models to bind our MongoDB data. It defines the structure that we would like to handle.
- Routes directory contains FastAPI routes definitions, where each route represents an endpoint that handles specific functions to serve data.
- Tests directory contains our unit tests for the application. This is especially a good practice to ensure the application functions as expected and avoid breaking our code while pushing to production.
- Requirements text file contains all our dependencies alongside their versions.
- Docker file and docker-compose files are used to store commands to assemble our image and set up a containerized application ensuring consistency.

To maintain a clean and scalable structure, we broke down our routes, tests and models as follow:

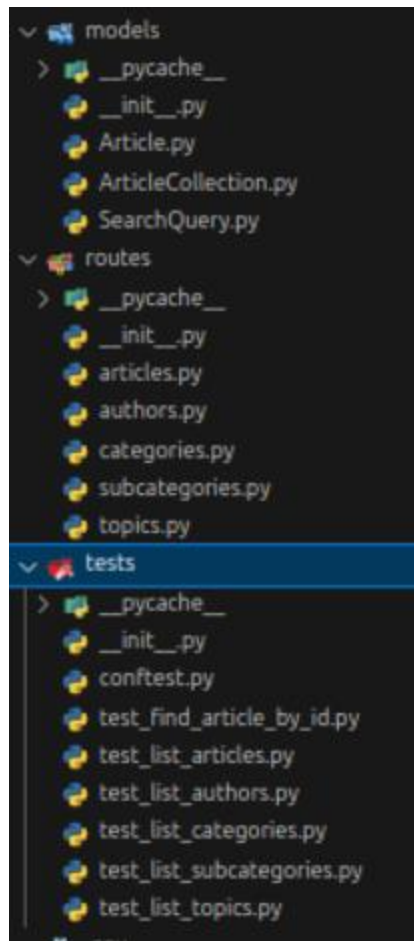


Figure 14: Backend detailed folder structure

- Models:
 - Article Binds data from MongoDB to our data model.
 - Article Collection represents a container for our articles. This is a good practice to ensure security rather than serving an array of objects directly from our endpoint.
 - Search Query represents a container for our full-text body search queries.
- Routes/Endpoints:
 - Articles: in this endpoint we try to serve all our data from an article point of view i.e.: each returned object will be an article instance. Its main endpoints are retrieved data, which could be filtered by a variety of fields. We also established another endpoint to gather data from a specific document ID. This will be useful for serving a specific document on our frontend.
 - Authors: in this endpoint we try to serve all of our authors related data, mainly: distinct authors list, number of articles posted by author and authors collaboration network i.e. The combination of links and nodes between authors and their binding group as needed (Productive author, irrelevant author etc.).

- Categories, Subcategories, Topics: in these endpoints we try to serve all our related data as follows: distinct values, number of articles grouped by the corresponding variable.

It is notable that we tried to minimize the load on the FastAPI by leveraging the power of MongoDB on querying and building pipelines to return the relevant data. Additional processing to data retrieved from MongoDB was applied to the network endpoint only as we needed to generate a big number of combinations of 2 between our authors lists.

Below is an example of how we retrieve our data while leveraging MongoDB:

```
pipeline = [
  {
    "$unwind": "$authors"
  },
  {
    "$group": {
      "_id": "$authors",
      "count": {"$sum": 1}
    }
  },
  {
    "$sort": {"count": -1}
  },
  {
    "$project": {
      "author": "$_id",
      "_id": 0, # exclude the grouping _id
      "count": 1
    }
  }
]
```

Figure 15: MongoDB Aggregation pipeline

The above pipeline explodes the authors arrays onto multiple records then groups them by authors and sums up the count of objects in each group. The resulting data is then sorted descending by count and projected to retrieve the needed data only. This reduces load for FastAPI (unneeded processing of each author's list, grouping and sorting) while increasing performance as our data is already indexed on MongoDB.

A description of data returned from our endpoints can be found on the [documentation.html](#) File on:

[News-Harbor/Software_Engineering at main · ABenazzou/News-Harbor \(github.com\)](#)

IV. Frontend Setup using React and D3.js

The frontend is developed using React, a popular JavaScript library for building user interfaces, and D3.js, a powerful tool for creating complex data visualizations. This combination allows for an

interactive, engaging, and informative user experience, effectively presenting the data-driven insights derived from the backend.

In this phase, the application's user interface was divided into several React components, such as navigation bars, content areas, and footers. This modular approach enhances readability and maintainability of the code. we started off by setting our project structure which came up as follow:

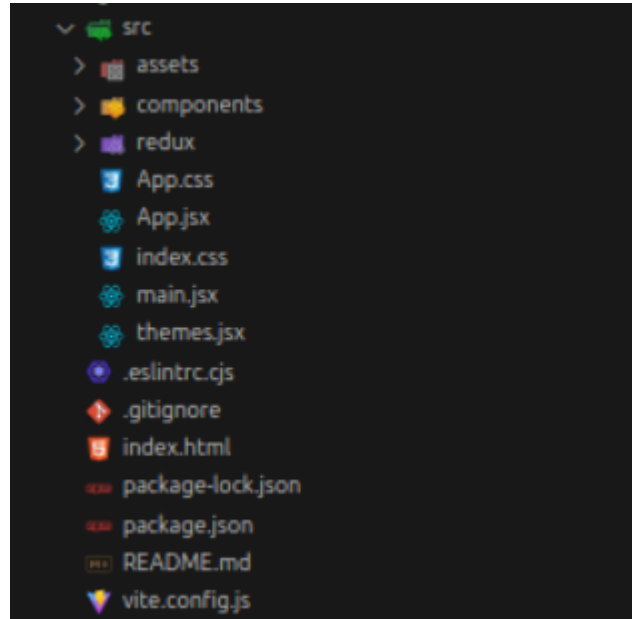


Figure 16: React project structure

- Any static data will be stored on the assets folder.
- Configuration will be done on vite.config.js to store the variable to our backend endpoint.
- All our rendered components (Header, Footer, Routed Body) will be stored under the components folder.
- As we incorporated dark/light themes, global state management was necessary to avoid prop drilling. For this purpose, redux was used and the corresponding slicers and reducers will be stored under this folder.
- App.jsx will engage our main layout and routes.
- Main.jsx represents our entry point for the application.
- Themes.jsx will store any global theme css variable we use to avoid the need of having to iterate the variables everywhere while the application scales (single source of truth).

Each component was designed to be reusable and independent as much as possible, following react best practices.

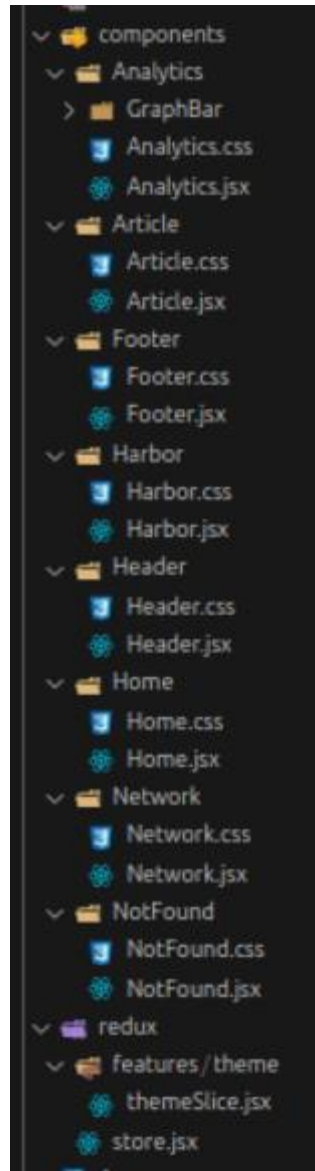


Figure 17: React components

- Analytics: This folder should contain all the visualizations that will be implemented inside the analytics tab. The main entry point will serve as a layout for our subcomponents:
 - GraphBar: The analytics tab contains a further subfolder which is a reusable GraphBar subcomponent written with D3.JS, incorporating specific data from our backend. This will be used because we intend to serve some basic analytics around our data such as Top authors, top topics etc.
- Article: This folder will serve our article data as intended to be read by the user.
- Header: A Navbar for our different sections of the app in addition to our theme toggle button.
- Footer: A placeholder Footer for copyrights and possibly more useful links in the future.
- Harbor: Our website main article searching feature. This will enable us to navigate the articles that are present in our backend and filter them using various criteria such as the

topics, the categories, the full text etc. To ensure a smooth experience the resulting articles will be opened in new tabs to not lose our user filters. A reset filter button is also provided to reduce manual cleanup.

- Home: A landing page where we present the purpose of our website using Carousels and represent the latest posted articles according to our backend.
- Network: A D3.js component to visualize the collaboration between authors so far on our website. This was done outside of the analytics tab to avoid cluttering our application and provide smoother navigation for our end user as needed.
- Not Found: A generic component for all the invalid routes for our web application.

Overall, the frontend setup utilizing React and D3.js demonstrates a thoughtful approach. The modular design and user-centric features contribute to the development of a robust and engaging application.

V. CI Pipelines Setup using GitHub Actions

As we hosted our application on a cloud compute instance, it was necessary to establish DevOps practices to reduce the manual deploy work and ensure the integrity of our repository without breaking our codebase. As such, each code push will trigger a corresponding CI pipeline (Backend or Frontend).

GitHub Actions allows the automation of workflows. We can create and test every commit or pull request to the repository, the workflows are stored under .github/workflows on the main Git repo entry.

Build Workflow:

- Trigger: On every push and pull request to the main branch.
- Actions:
 - Check out the code.
 - Installs dependencies.
 - Runs the build process to ensure code compiles without errors.
 - [Specific to Backend] Deploys a temporary docker image of our backend and runs the tests, if any of them fails, the deploy will be halted, thus not breaking our production.
 - Deploy the build code to the correspondent directory on the hosting server and serve it under Docker (Backend Containerized FastAPI) or PM2 (Frontend process manager for node.js).

The setup of CI pipelines using GitHub Actions has significantly streamlined the development process of the News Harbor project. This automation not only ensures the reliability and stability of the application but also enhances the efficiency and collaboration of the development team if any is planned in the future.

VI. Conclusion

As we reach the culmination of the documentation for the "News Harbor" project, it is crucial to reflect on the journey we have undertaken, the milestones achieved, and the horizons yet to explore.

Summary of Achievements

- **Successful Data Scraping and Processing:** Through a combination of tools like Selenium, Requests, and Apache Airflow, we have effectively automated the collection and preprocessing of news data from the BBC news website.
- **Robust Data Cleaning and Analysis:** Implementing thorough data cleaning techniques and insightful analysis, we have transformed raw data into a valuable resource for information and decision-making.
- **Innovative Backend and Frontend Development:** By utilizing FastAPI and React along with D3.js, the project boasts of a powerful backend and a dynamic, user-friendly frontend.
- **Efficient CI/CD Pipelines:** The integration of Continuous Integration and Continuous Deployment pipelines using GitHub Actions has significantly streamlined our development and deployment processes, ensuring consistent code quality and operational efficiency.

Challenges and Overcoming Them

Throughout this journey, we faced various challenges such as optimizing data scraping methods, ensuring data quality, and implementing effective data visualization strategies. Each of these challenges was an opportunity for learning and growth. By adopting a flexible and innovative approach, we overcome these hurdles, enhancing the project's robustness and reliability.

Reflections and Learnings

This project has been a testament to the power of combining different technologies and methodologies to achieve a cohesive and efficient system. It underscored the importance of continuous learning, adaptability, and the value of a well-thought-out design and architectural planning.

As we look to the future, the "News Harbor" project holds the potential for further enhancements and expansions. Possible future developments could include:

- **Integration with Additional Data Sources:** Expanding the scope to include more news sources for a more comprehensive data set.
- **Advanced Analytics Features:** Implementing more sophisticated analytics tools with D3.js as it was not leveraged appropriately due to the learning curve and time frame.
- **Enhanced User Personalization:** Developing more personalized user experiences based on user behavior and preferences.

- Authentication System for additional features: Provide JWT based authentication system which allows users to have advantage over guests such as saving their filters, customizing their analytics, etc.

In conclusion, the "News Harbor" project stands as a robust and dynamic solution, adept at navigating the vast sea of online information. It has been a pleasure to work on such a project.